ДОДАТОК А

Програмний код

BoundingBox

```
//
//  BoundingBox.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 11.11.2019.
//  Copyright © 2018 K.S. All rights reserved.
//

import Foundation
import UIKit

class BoundingBox {

  let shapeLayer: CAShapeLayer
  let textLayer: CATextLayer

  init() {
    shapeLayer = CAShapeLayer()
    shapeLayer.fillColor = UIColor.clear.cgColor
    shapeLayer.lineWidth = 4
    shapeLayer.isHidden = true

    textLayer = CATextLayer()
    textLayer.foregroundColor = UIColor.black.cgColor
    textLayer.isHidden = true
    textLayer.contentsScale = UIScreen.main.scale
    textLayer.fontSize = 14
    textLayer.font = UIFont(name: "Avenir", size: textLayer.fontSize)
    textLayer.alignmentMode = kCAAlignmentCenter
  }

  func addToLayer(_ parent: CALayer) {
    parent.addSublayer(shapeLayer)
    parent.addSublayer(textLayer)
  }

  func show(frame: CGRect, label: String, color: UIColor) {
    CATransaction.setDisableActions(true)

    let path = UIBezierPath(rect: frame)
    shapeLayer.path = path.cgPath
    shapeLayer.strokeColor = color.cgColor
```

```
    shapeLayer.isHidden = false

    textLayer.string = label
    textLayer.backgroundColor = color.cgColor
    textLayer.isHidden = false

    let attributes = [
      NSAttributedString.Key.font: textLayer.font as Any
    ]

    let textRect = label.boundingRect(with: CGSize(width: 400, height: 100),
                                      options: .truncatesLastVisibleLine,
                                      attributes: attributes, context: nil)
    let textSize = CGSize(width: textRect.width + 12, height: textRect.height)
    let textOrigin = CGPoint(x: frame.origin.x - 2, y: frame.origin.y - textSize.height)
    textLayer.frame = CGRect(origin: textOrigin, size: textSize)
  }

  func hide() {
    shapeLayer.isHidden = true
    textLayer.isHidden = true
  }
}
```

DVideoCapture

```
//
//  DVideoCapture.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 11.11.2019.
//  Copyright © 2018 K.S. All rights reserved.
//

import UIKit
import AVFoundation
import CoreVideo

public protocol DVideoCaptureDelegate: class {
  func videoCapture(_ capture: DVideoCapture, didCaptureVideoFrame: CVPixelBuffer?,
timestamp: CMTime)
}

public class DVideoCapture: NSObject {
  public var previewLayer: AVCaptureVideoPreviewLayer?
  public weak var delegate: DVideoCaptureDelegate?
  public var desiredFrameRate = 30

  let captureSession = AVCaptureSession()
  let videoOutput = AVCaptureVideoDataOutput()
  let queue = DispatchQueue(label: "net.machinethink.camera-queue")

  public func setUp(sessionPreset: AVCaptureSession.Preset = .medium,
                    completion: @escaping (Bool) -> Void) {
    queue.async {
      let success = self.setUpCamera(sessionPreset: sessionPreset)
      DispatchQueue.main.async {
        completion(success)
      }
    }
  }

  func setUpCamera(sessionPreset: AVCaptureSession.Preset) -> Bool {
    captureSession.beginConfiguration()
    captureSession.sessionPreset = sessionPreset

    guard let captureDevice = AVCaptureDevice.default(for: AVMediaType.video) else {
      print("Error: no video devices available")
      return false
    }

    guard let videoInput = try? AVCaptureDeviceInput(device: captureDevice) else {
      print("Error: could not create AVCaptureDeviceInput")
      return false
```

```
    }

    if captureSession.canAddInput(videoInput) {
      captureSession.addInput(videoInput)
    }

    let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    previewLayer.videoGravity = AVLayerVideoGravity.resizeAspect
    previewLayer.connection?.videoOrientation = .portrait
    self.previewLayer = previewLayer

    let settings: [String : Any] = [
      kCVPixelBufferPixelFormatTypeKey as String: NSNumber(value:
kCVPixelFormatType_32BGRA),
    ]

    videoOutput.videoSettings = settings
    videoOutput.alwaysDiscardsLateVideoFrames = true
    videoOutput.setSampleBufferDelegate(self, queue: queue)
    if captureSession.canAddOutput(videoOutput) {
      captureSession.addOutput(videoOutput)
    }

    // We want the buffers to be in portrait orientation otherwise they are
    // rotated by 90 degrees. Need to set this _after_ addOutput()!
    videoOutput.connection(with: AVMediaType.video)?.videoOrientation = .portrait

    // Based on code from https://github.com/dokun1/Lumina/
    let activeDimensions = CMVideoFormatDescript-
onGetDimensions(captureDevice.activeFormat.formatDescription)
    for vFormat in captureDevice.formats {
      let dimensions = CMVideoFormatDescriptionGetDimensions(vFormat.formatDescription)
      let ranges = vFormat.videoSupportedFrameRateRanges as [AVFrameRateRange]
      if let frameRate = ranges.first,
        frameRate.maxFrameRate >= Float64(desiredFrameRate) &&
          frameRate.minFrameRate <= Float64(desiredFrameRate) &&
          activeDimensions.width == dimensions.width &&
          activeDimensions.height == dimensions.height &&
          CMFormatDescriptionGetMediaSubType(vFormat.formatDescription) == 875704422 { //
meant for full range 420f
        do {
          try captureDevice.lockForConfiguration()
          captureDevice.activeFormat = vFormat as AVCaptureDevice.Format
          captureDevice.activeVideoMinFrameDuration = CMTimeMake(1, Int32(desiredFrameRate))
          captureDevice.activeVideoMaxFrameDuration = CMTimeMake(1, Int32(desiredFrameRate))
          captureDevice.unlockForConfiguration()
          break
        } catch {
          continue
        }
      }
    }
    print("Camera format:", captureDevice.activeFormat)
```

```
    captureSession.commitConfiguration()
    return true
  }

  public func start() {
    if !captureSession.isRunning {
      captureSession.startRunning()
    }
  }

  public func stop() {
    if captureSession.isRunning {
      captureSession.stopRunning()
    }
  }
}

extension DVideoCapture: AVCaptureVideoDataOutputSampleBufferDelegate {
  public func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer:
CMSampleBuffer, from connection: AVCaptureConnection) {
    let timestamp = CMSampleBufferGetPresentationTimeStamp(sampleBuffer)
    let imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
    delegate?.videoCapture(self, didCaptureVideoFrame: imageBuffer, timestamp: timestamp)
  }

  public func captureOutput(_ output: AVCaptureOutput, didDrop sampleBuffer: CMSampleBuffer,
from connection: AVCaptureConnection) {
  }
}
```

PredictionManager

```
//
//  PredictionManager.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 9.10.2019.
//  Copyright © 2018 K.S. All rights reserved.
//

import Foundation
import UIKit
import CoreML

class PredictionManager {

  public static let inputWidth = 416
  public static let inputHeight = 416
  public static let maxBoundingBoxes = 8

  let confidenceThreshold: Float = 0.3
  let iouThreshold: Float = 0.5

  struct Prediction {
    let classIndex: Int
    let score: Float
    let rect: CGRect
  }

  let model = DriverAssist()

  public init() { }

  public func predict(image: CVPixelBuffer) -> [Prediction]? {
    if let output = try? model.prediction(image: image) {
      return computeBoundingBoxes(features: output.grid)
    } else {
      return nil
    }
  }

  public func computeBoundingBoxes(features: MLMultiArray) -> [Prediction] {
    assert(features.count == 125*13*13)

    var predictions = [Prediction]()

    let blockSize: Float = 32
    let gridHeight = 13
    let gridWidth = 13
    let boxesPerCell = 5
    let numClasses = 20
```

```
    let featurePointer = UnsafeMutablePointer<Double>(OpaquePointer(features.dataPointer))
    let channelStride = features.strides[0].intValue
    let yStride = features.strides[1].intValue
    let xStride = features.strides[2].intValue

    @inline(__always) func offset(_ channel: Int, _ x: Int, _ y: Int) -> Int {
      return channel*channelStride + y*yStride + x*xStride
    }

    for cy in 0..<gridHeight {
      for cx in 0..<gridWidth {
        for b in 0..<boxesPerCell {
          let channel = b*(numClasses + 5)
          let tx = Float(featurePointer[offset(channel    , cx, cy)])
          let ty = Float(featurePointer[offset(channel + 1, cx, cy)])
          let tw = Float(featurePointer[offset(channel + 2, cx, cy)])
          let th = Float(featurePointer[offset(channel + 3, cx, cy)])
          let tc = Float(featurePointer[offset(channel + 4, cx, cy)])
          let x = (Float(cx) + sigmoid(tx)) * blockSize
          let y = (Float(cy) + sigmoid(ty)) * blockSize
          let w = exp(tw) * anchors[2*b    ] * blockSize
          let h = exp(th) * anchors[2*b + 1] * blockSize

          let confidence = sigmoid(tc)

          var classes = [Float](repeating: 0, count: numClasses)
          for c in 0..<numClasses {
            classes[c] = Float(featurePointer[offset(channel + 5 + c, cx, cy)])
          }
          classes = softmax(classes)

          let (detectedClass, bestClassScore) = classes.argmax()
          let confidenceInClass = bestClassScore * confidence

          if confidenceInClass > confidenceThreshold {
            let rect = CGRect(x: CGFloat(x - w/2), y: CGFloat(y - h/2),
                              width: CGFloat(w), height: CGFloat(h))

            let prediction = Prediction(classIndex: detectedClass,
                                        score: confidenceInClass,
                                        rect: rect)
            predictions.append(prediction)
          }
        }
      }
    }

    return nonMaxSuppression(boxes: predictions, limit: PredictionManager.maxBoundingBoxes,
threshold: iouThreshold)
  }
}
```

StartViewController

```
//
//  StartViewController.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 18.04.2018.
//  Copyright © 2018 K.S. All rights reserved.
//

import UIKit

class StartViewController: UIViewController, UITextFieldDelegate {

  override func viewDidLoad() {
    super.viewDidLoad()
  }

  @IBAction func startVision(_ sender: Any) {
    self.performSegue(withIdentifier: "openVisionML", sender: "vision")
  }

  @IBAction func startML(_ sender: Any) {
    self.performSegue(withIdentifier: "openVisionML", sender: "ml")
  }

  @IBAction func startDriverAssist(_ sender: Any) {
    self.performSegue(withIdentifier: "openDriverAssist", sender: "vision")
  }

  func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    self.view.endEditing(true)
    return false
  }

  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "openVisionML" {
      if let vc = segue.destination as? MainViewController {
        vc.type = sender as? String ?? ""
      }
    }
  }
}
```

MainViewController

```
//
//  MainViewController.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 18.04.2018.
//  Copyright © 2018 K.S. All rights reserved.
//

import UIKit
import CoreMedia
import Vision

class MainViewController: UIViewController, UIImagePickerControllerDelegate {

  @IBOutlet private weak var predictLabel: UILabel!
  @IBOutlet private weak var trustedLabel: UILabel!
  @IBOutlet private weak var previewView: UIView!
  @IBOutlet private weak var focusedView: UIView!

  public var type: String = ""
  public var eps: String = "0.7"

  private let inceptionv3model = Inceptionv3()
  private var videoCapture: VideoCapture!
  private var requests = [VNRequest]()

  override func viewDidLoad() {
    super.viewDidLoad()

    self.navigationItem.title = self.type + " OFF!"

    self.focusedView.layer.cornerRadius = 10
    self.predictLabel.layer.cornerRadius = 10
    self.previewView.layer.cornerRadius = 10

    let spec = VideoSpec(fps: 5, size: CGSize(width: 299, height: 299))

    videoCapture = VideoCapture(cameraType: .back,
                                preferredSpec: spec,
                                previewContainer: previewView.layer)

    videoCapture.imageBufferHandler = {[unowned self] (imageBuffer) in

      DispatchQueue.main.async {
        self.focusedView.backgroundColor = UIColor.clear
        self.focusedView.layer.borderColor = UIColor.green.cgColor
        self.focusedView.layer.borderWidth = 2.0
        self.focusedView.isHidden = true
        self.trustedLabel.text = ""
      }
```

```
    if self.type == "vision" {
      self.setupVision()
      print(self.type + " recognition...")
      self.handleImageBufferWithVision(imageBuffer: imageBuffer)
    } else {
      self.navigationItem.title = self.type + " ON"
      print(self.type + " recognition...")
      self.handleImageBufferWithCoreML(imageBuffer: imageBuffer)
    }}
  }

  func handleImageBufferWithCoreML(imageBuffer: CMSampleBuffer) {
    guard let pixelBuffer = CMSampleBufferGetImageBuffer(imageBuffer) else {
      return
    }
    do {
      let prediction = try self.inceptionv3model.prediction(image: self.resize(pixelBuffer:
pixelBuffer)!)
      DispatchQueue.main.async {
        if let prob = prediction.classLabelProbs[prediction.classLabel] {
          self.predictLabel.text = "\(prediction.classLabel) \(String(describing: prob))"
          print("%" + String(describing: prob))
          let percentD = Float(prob) as Float
          let epsD = Float(self.eps)!

          if percentD > epsD {
            self.videoCapture.stopCapture()
            DispatchQueue.main.asyncAfter(deadline: .now() + 0.3, execute: {
              self.predictionCorrect(pred: "\(prediction.classLabel)")
            })
          }
        }
      }
    }
    catch let error as NSError {
      fatalError("Unexpected error ocurred: \(error.localizedDescription).")
    }
  }

  func predictionCorrect(pred : String) {
    DispatchQueue.main.async {
      self.focusedView.isHidden = false
      self.trustedLabel.isHidden = false
      self.trustedLabel.text = "It's a " + pred
      self.trustedLabel.layer.cornerRadius = 30
    }
  }

  func handleImageBufferWithVision(imageBuffer: CMSampleBuffer) {
    guard let pixelBuffer = CMSampleBufferGetImageBuffer(imageBuffer) else {
      return
    }
```

```
    var requestOptions: [VNImageOption : Any] = [:]
    if let cameraIntrinsicData = CMGetAttachment(imageBuffer,
kCMSampleBufferAttachmentKey_CameraIntrinsicMatrix, nil) {
      requestOptions = [.cameraIntrinsics:cameraIntrinsicData]
    }

    let imageRequestHandler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation:
CGImagePropertyOrientation(rawValue: UInt32(self.exifOrientationFromDeviceOrientation))!,
options: requestOptions)
    do {
      try imageRequestHandler.perform(self.requests)
    } catch {
      print(error)
    }
  }

  func setupVision() {
    self.navigationItem.title = self.type + " ON"

    guard let visionModel = try? VNCoreMLModel(for: inceptionv3model.model) else {
      fatalError("can't load Vision ML model")
    }
    let classificationRequest = VNCoreMLRequest(model: visionModel) { (request: VNRequest,
error: Error?) in
      guard let observations = request.results else {
        print("no results:\(error!)")
        return
      }
      let classifications = observations[0...4]
        .compactMap({ $0 as? VNClassificationObservation })
        .filter({ $0.confidence > 0.2 })
        .map({ "\($0.identifier) \($0.confidence)" })
      DispatchQueue.main.async {
        self.predictLabel.text = classifications.joined(separator: "\n")
      }
    }
    classificationRequest.imageCropAndScaleOption = VNImageCropAndScaleOption.centerCrop

    self.requests = [classificationRequest]
  }
  /// only support back camera
  var exifOrientationFromDeviceOrientation: Int32 {
    let exifOrientation: DeviceOrientation
    enum DeviceOrientation: Int32 {
      case top0ColLeft = 1
      case top0ColRight = 2
      case bottom0ColRight = 3
      case bottom0ColLeft = 4
      case left0ColTop = 5
      case right0ColTop = 6
      case right0ColBottom = 7
      case left0ColBottom = 8
```

```
    }
    switch UIDevice.current.orientation {
    case .portraitUpsideDown:
      exifOrientation = .left0ColBottom
    case .landscapeLeft:
      exifOrientation = .top0ColLeft
    case .landscapeRight:
      exifOrientation = .bottom0ColRight
    default:
      exifOrientation = .right0ColTop
    }
    return exifOrientation.rawValue
  }
  func resize(pixelBuffer: CVPixelBuffer) -> CVPixelBuffer? {
    let imageSide = 299
    var ciImage = CIImage(cvPixelBuffer: pixelBuffer, options: nil)
    let transform = CGAffineTransform(scaleX: CGFloat(imageSide) /
CGFloat(CVPixelBufferGetWidth(pixelBuffer)), y: CGFloat(imageSide) /
CGFloat(CVPixelBufferGetHeight(pixelBuffer)))
    ciImage = ciImage.transformed(by: transform).cropped(to: CGRect(x: 0, y: 0, width: imageSide,
height: imageSide))
    let ciContext = CIContext()
    var resizeBuffer: CVPixelBuffer?
    CVPixelBufferCreate(kCFAllocatorDefault, imageSide, imageSide,
CVPixelBufferGetPixelFormatType(pixelBuffer), nil, &resizeBuffer)
    ciContext.render(ciImage, to: resizeBuffer!)
    return resizeBuffer
  }
  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    guard let videoCapture = videoCapture else {return}
    videoCapture.startCapture()
  }
  override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    guard let videoCapture = videoCapture else {return}
    videoCapture.resizePreview()
  }

  override func viewWillDisappear(_ animated: Bool) {
    guard let videoCapture = videoCapture else {return}
    videoCapture.stopCapture()

    navigationController?.setNavigationBarHidden(false, animated: true)
    super.viewWillDisappear(animated)
  }
}
```

DriverAssistViewController

```
//
//  DriverAssistViewController.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 11.11.2019.
//  Copyright © 2018 K.S. All rights reserved.
//

import UIKit
import Vision
import AVFoundation
import CoreMedia

class DriverAssistViewController: UIViewController {

  @IBOutlet weak var videoPreview: UIView!
  @IBOutlet weak var timeLabel: UILabel!

  // true: use Vision to drive Core ML, false: use plain Core ML
  let useVision = true

  // Disable this to see the energy impact of just running the neural net,
  // otherwise it also counts the GPU activity of drawing the bounding boxes.
  let drawBoundingBoxes = true

  // How many predictions we can do concurrently.
  static let maxInflightBuffers = 3

  let predictionManager = PredictionManager()

  var videoCapture: DVideoCapture!
  var requests = [VNCoreMLRequest]()
  var startTimes: [CFTimeInterval] = []

  var boundingBoxes = [BoundingBox]()
  var colors: [UIColor] = []

  let ciContext = CIContext()
  var resizedPixelBuffers: [CVPixelBuffer?] = []

  var framesDone = 0
  var frameCapturingStartTime = CACurrentMediaTime()

  var inflightBuffer = 0
  let semaphore = DispatchSemaphore(value: DriverAssistViewController.maxInflightBuffers)

  override func viewDidLoad() {
    super.viewDidLoad()

    timeLabel.text = ""
```

```
    setUpBoundingBoxes()
    setUpCoreImage()
    setUpVision()
    setUpCamera()

    frameCapturingStartTime = CACurrentMediaTime()
  }

  // MARK: - Initialization

  func setUpBoundingBoxes() {
    for _ in 0..<PredictionManager.maxBoundingBoxes {
      boundingBoxes.append(BoundingBox())
    }

    // Make colors for the bounding boxes. There is one color for each class,
    // 20 classes in total.
    for r: CGFloat in [0.2, 0.4, 0.6, 0.8, 1.0] {
      for g: CGFloat in [0.3, 0.7] {
        for b: CGFloat in [0.4, 0.8] {
          let color = UIColor(red: r, green: g, blue: b, alpha: 1)
          colors.append(color)
        }
      }
    }
  }

  func setUpCoreImage() {
    for _ in 0..<DriverAssistViewController.maxInflightBuffers {
      var resizedPixelBuffer: CVPixelBuffer?
      let status = CVPixelBufferCreate(nil, PredictionManager.inputWidth,
PredictionManager.inputHeight,
                                       kCVPixelFormatType_32BGRA, nil,
                                       &resizedPixelBuffer)
      if status != kCVReturnSuccess {
        print("Error: could not create resized pixel buffer", status)
      }
      resizedPixelBuffers.append(resizedPixelBuffer)
    }
  }

  func setUpVision() {
    guard let visionModel = try? VNCoreMLModel(for: predictionManager.model.model) else {
      print("Error: could not create Vision model")
      return
    }
    for _ in 0..<DriverAssistViewController.maxInflightBuffers {
      let request = VNCoreMLRequest(model: visionModel, completionHandler:
visionRequestDidComplete)
      request.imageCropAndScaleOption = .scaleFill
      requests.append(request)
    }
```

```
}

func setUpCamera() {
  videoCapture = DVideoCapture()
  videoCapture.delegate = self
  videoCapture.desiredFrameRate = 60
  videoCapture.setUp(sessionPreset: AVCaptureSession.Preset.hd1280x720) { success in
    if success {
      // Add the video preview into the UI.
      if let previewLayer = self.videoCapture.previewLayer {
        self.videoPreview.layer.addSublayer(previewLayer)
        self.resizePreviewLayer()
      }
      // Add the bounding box layers to the UI, on top of the video preview.
      for box in self.boundingBoxes {
        box.addToLayer(self.videoPreview.layer)
      }
      // Once everything is set up, we can start capturing live video.
      self.videoCapture.start()
    }
  }
}

override func viewWillLayoutSubviews() {
  super.viewWillLayoutSubviews()

  resizePreviewLayer()
}

override var preferredStatusBarStyle: UIStatusBarStyle {
  return .lightContent
}

func resizePreviewLayer() {
  videoCapture.previewLayer?.frame = videoPreview.bounds
}

// MARK: - Doing inference

func predict(image: UIImage) {
  if let pixelBuffer = image.pixelBuffer(width: PredictionManager.inputWidth,
                                         height: PredictionManager.inputHeight) {
    predict(pixelBuffer: pixelBuffer, inflightIndex: 0)
  }
}

func predict(pixelBuffer: CVPixelBuffer, inflightIndex: Int) {
  // Measure how long it takes to predict a single video frame.
  let startTime = CACurrentMediaTime()
  // Resize the input with Core Image to 416x416.
  if let resizedPixelBuffer = resizedPixelBuffers[inflightIndex] {
    let ciImage = CIImage(cvPixelBuffer: pixelBuffer)
    let sx = CGFloat(PredictionManager.inputWidth) /
```

```
CGFloat(CVPixelBufferGetWidth(pixelBuffer))
      let sy = CGFloat(PredictionManager.inputHeight) /
CGFloat(CVPixelBufferGetHeight(pixelBuffer))
      let scaleTransform = CGAffineTransform(scaleX: sx, y: sy)
      let scaledImage = ciImage.transformed(by: scaleTransform)
      ciContext.render(scaledImage, to: resizedPixelBuffer)
      // Give the resized input to our model.
      if let boundingBoxes = predictionManager.predict(image: resizedPixelBuffer) {
        let elapsed = CACurrentMediaTime() - startTime
        showOnMainThread(boundingBoxes, elapsed)
      }
    }
    self.semaphore.signal()
  }

  func predictUsingVision(pixelBuffer: CVPixelBuffer, inflightIndex: Int) {
    startTimes.append(CACurrentMediaTime())
    // Vision will automatically resize the input image.
    let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer)
    let request = requests[inflightIndex]
    DispatchQueue.global().async {
      try? handler.perform([request])
    }
  }

  func visionRequestDidComplete(request: VNRequest, error: Error?) {
    if let observations = request.results as? [VNCoreMLFeatureValueObservation],
      let features = observations.first?.featureValue.multiArrayValue {
      let boundingBoxes = predictionManager.computeBoundingBoxes(features: features)
      let elapsed = CACurrentMediaTime() - startTimes.remove(at: 0)
      showOnMainThread(boundingBoxes, elapsed)
    }
    self.semaphore.signal()
  }

  func showOnMainThread(_ boundingBoxes: [PredictionManager.Prediction], _ elapsed:
CFTimeInterval) {
    if drawBoundingBoxes {
      DispatchQueue.main.async {
        self.show(predictions: boundingBoxes)
        let fps = self.measureFPS()
        self.timeLabel.text = String(format: "Recognition - %.2f FPS", fps)
      }
    }
  }

  func measureFPS() -> Double {
    // Measure how many frames were actually delivered per second.
    framesDone += 1
    let frameCapturingElapsed = CACurrentMediaTime() - frameCapturingStartTime
    let currentFPSDelivered = Double(framesDone) / frameCapturingElapsed
    if frameCapturingElapsed > 1 {
      framesDone = 0
```

```
      frameCapturingStartTime = CACurrentMediaTime()
    }
    return currentFPSDelivered
  }

  func show(predictions: [PredictionManager.Prediction]) {
    for i in 0..<boundingBoxes.count {
      if i < predictions.count {
        let prediction = predictions[i]
        // Car or Person
        if prediction.classIndex == 6 || prediction.classIndex == 14 {
          let width = view.bounds.width
          let height = width * 16 / 9
          let scaleX = width / CGFloat(PredictionManager.inputWidth)
          let scaleY = height / CGFloat(PredictionManager.inputHeight)
          let top = (view.bounds.height - height) / 2

          var rect = prediction.rect
          rect.origin.x *= scaleX
          rect.origin.y *= scaleY
          rect.origin.y += top
          rect.size.width *= scaleX
          rect.size.height *= scaleY

          var color = UIColor.white
          // Car
          if prediction.classIndex == 6 {
            color = UIColor(red:0.39, green:0.58, blue:0.93, alpha:1.0)
          }
          // Person
          if prediction.classIndex == 14 {
            color = UIColor(red:1.00, green:0.27, blue:0.27, alpha:1.0)
          }

          let label = String(format: "%@ %.1f", labels[prediction.classIndex], prediction.score * 100)
          boundingBoxes[i].show(frame: rect, label: label, color: color)
        }
      } else {
        boundingBoxes[i].hide()
      }
    }
  }
}

extension DriverAssistViewController: DVideoCaptureDelegate {

  func videoCapture(_ capture: DVideoCapture, didCaptureVideoFrame pixelBuffer:
CVPixelBuffer?, timestamp: CMTime) {
    if let pixelBuffer = pixelBuffer {
      semaphore.wait()
      let inflightIndex = inflightBuffer
      inflightBuffer += 1
      if inflightBuffer >= DriverAssistViewController.maxInflightBuffers {
```

```
        inflightBuffer = 0
      }
      if useVision {
        // This method should always be called from the same thread!
        // Ain't nobody likes race conditions and crashes.
        self.predictUsingVision(pixelBuffer: pixelBuffer, inflightIndex: inflightIndex)
      } else {
        // For better throughput, perform the prediction on a concurrent
        // background queue instead of on the serial VideoCapture queue.
        DispatchQueue.global().async {
          self.predict(pixelBuffer: pixelBuffer, inflightIndex: inflightIndex)
        }
      }
    }
  }
}
```

UIImage+CVPixelBuffer

```
//
//  UIImage+CVPixelBuffer.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 10.10.2018.
//  Copyright © 2018 K.S. All rights reserved.
//

import UIKit

extension UIImage {
  public func pixelBuffer(width: Int, height: Int) -> CVPixelBuffer? {
    var maybePixelBuffer: CVPixelBuffer?
    let attrs = [kCVPixelBufferCGImageCompatibilityKey: kCFBooleanTrue,
                 kCVPixelBufferCGBitmapContextCompatibilityKey: kCFBooleanTrue]
    let status = CVPixelBufferCreate(kCFAllocatorDefault,
                                     Int(width),
                                     Int(height),
                                     kCVPixelFormatType_32ARGB,
                                     attrs as CFDictionary,
                                     &maybePixelBuffer)

    guard status == kCVReturnSuccess, let pixelBuffer = maybePixelBuffer else {
      return nil
    }

    CVPixelBufferLockBaseAddress(pixelBuffer, CVPixelBufferLockFlags(rawValue: 0))
    let pixelData = CVPixelBufferGetBaseAddress(pixelBuffer)

    guard let context = CGContext(data: pixelData,
                                  width: Int(width),
                                  height: Int(height),
                                  bitsPerComponent: 8,
                                  bytesPerRow: CVPixelBufferGetBytesPerRow(pixelBuffer),
                                  space: CGColorSpaceCreateDeviceRGB(),
                                  bitmapInfo: CGImageAlphaInfo.noneSkipFirst.rawValue)
    else {
      return nil
    }
    context.translateBy(x: 0, y: CGFloat(height))
    context.scaleBy(x: 1, y: -1)
    UIGraphicsPushContext(context)
    self.draw(in: CGRect(x: 0, y: 0, width: width, height: height))
    UIGraphicsPopContext()
    CVPixelBufferUnlockBaseAddress(pixelBuffer, CVPixelBufferLockFlags(rawValue: 0))
    return pixelBuffer
  }
}
```

CVPixelBuffer+Helpers

```
//
//  CVPixelBuffer+Helpers.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 10.10.2018.
//  Copyright © 2018 K.S. All rights reserved.
//

import Foundation
import Accelerate

func resizePixelBuffer(_ srcPixelBuffer: CVPixelBuffer,
                       cropX: Int,
                       cropY: Int,
                       cropWidth: Int,
                       cropHeight: Int,
                       scaleWidth: Int,
                       scaleHeight: Int) -> CVPixelBuffer? {

  CVPixelBufferLockBaseAddress(srcPixelBuffer, CVPixelBufferLockFlags(rawValue: 0))
  guard let srcData = CVPixelBufferGetBaseAddress(srcPixelBuffer) else {
    print("Error: could not get pixel buffer base address")
    return nil
  }
  let srcBytesPerRow = CVPixelBufferGetBytesPerRow(srcPixelBuffer)
  let offset = cropY*srcBytesPerRow + cropX*4
  var srcBuffer = vImage_Buffer(data: srcData.advanced(by: offset),
                                height: vImagePixelCount(cropHeight),
                                width: vImagePixelCount(cropWidth),
                                rowBytes: srcBytesPerRow)

  let destBytesPerRow = scaleWidth*4
  guard let destData = malloc(scaleHeight*destBytesPerRow) else {
    print("Error: out of memory")
    return nil
  }
  var destBuffer = vImage_Buffer(data: destData,
                                 height: vImagePixelCount(scaleHeight),
                                 width: vImagePixelCount(scaleWidth),
                                 rowBytes: destBytesPerRow)

  let error = vImageScale_ARGB8888(&srcBuffer, &destBuffer, nil, vImage_Flags(0))
  CVPixelBufferUnlockBaseAddress(srcPixelBuffer, CVPixelBufferLockFlags(rawValue: 0))
  if error != kvImageNoError {
    print("Error:", error)
    free(destData)
    return nil
  }

  let releaseCallback: CVPixelBufferReleaseBytesCallback = { _, ptr in
```

```
    if let ptr = ptr {
      free(UnsafeMutableRawPointer(mutating: ptr))
    }
  }

  let pixelFormat = CVPixelBufferGetPixelFormatType(srcPixelBuffer)
  var dstPixelBuffer: CVPixelBuffer?
  let status = CVPixelBufferCreateWithBytes(nil, scaleWidth, scaleHeight,
                                    pixelFormat, destData,
                                    destBytesPerRow, releaseCallback,
                                    nil, nil, &dstPixelBuffer)
  if status != kCVReturnSuccess {
    print("Error: could not create new pixel buffer")
    free(destData)
    return nil
  }
  return dstPixelBuffer
}

func resizePixelBuffer(_ pixelBuffer: CVPixelBuffer,
                       width: Int, height: Int) -> CVPixelBuffer? {
  return resizePixelBuffer(pixelBuffer, cropX: 0, cropY: 0,
                           cropWidth: CVPixelBufferGetWidth(pixelBuffer),
                           cropHeight: CVPixelBufferGetHeight(pixelBuffer),
                           scaleWidth: width, scaleHeight: height)
}
```

AVCaptureDevice+Extension

```
//
//  AVCaptureDevice+Extension.swift
//  DriverAssist
//
//  Created by Konstantin Stolyarenko on 10.10.2018.
//  Copyright © 2018 K.S. All rights reserved.
//

import AVFoundation

extension AVCaptureDevice {
  private func availableFormatsFor(preferredFps: Float64) -> [AVCaptureDevice.Format] {
    var availableFormats: [AVCaptureDevice.Format] = []
    for format in formats {
      let ranges = format.videoSupportedFrameRateRanges
      for range in ranges where range.minFrameRate <= preferredFps && preferredFps <=
range.maxFrameRate {
        availableFormats.append(format)
      }
    }
    return availableFormats
  }

  private func formatWithHighestResolution(_ availableFormats: [AVCaptureDevice.Format]) ->
AVCaptureDevice.Format? {
    var maxWidth: Int32 = 0
    var selectedFormat: AVCaptureDevice.Format?
    for format in availableFormats {
      let desc = format.formatDescription
      let dimensions = CMVideoFormatDescriptionGetDimensions(desc)
      let width = dimensions.width
      if width >= maxWidth {
        maxWidth = width
        selectedFormat = format
      }
    }
    return selectedFormat
  }

  private func formatFor(preferredSize: CGSize, availableFormats: [AVCaptureDevice.Format]) ->
AVCaptureDevice.Format? {
    for format in availableFormats {
      let desc = format.formatDescription
      let dimensions = CMVideoFormatDescriptionGetDimensions(desc)
      if dimensions.width >= Int32(preferredSize.width) && dimensions.height >=
Int32(preferredSize.height) {
        return format
      }
    }
    return nil
```

```
  }

  func updateFormatWithPreferredVideoSpec(preferredSpec: VideoSpec) {
    let availableFormats: [AVCaptureDevice.Format]
    if let preferredFps = preferredSpec.fps {
      availableFormats = availableFormatsFor(preferredFps: Float64(preferredFps))
    } else {
      availableFormats = formats
    }

    var selectedFormat: AVCaptureDevice.Format?
    if let preferredSize = preferredSpec.size {
      selectedFormat = formatFor(preferredSize: preferredSize, availableFormats: availableFormats)
    } else {
      selectedFormat = formatWithHighestResolution(availableFormats)
    }
    print("selected format: \(String(describing: selectedFormat))")

    if let selectedFormat = selectedFormat {
      do {
        try lockForConfiguration()
      } catch {
        fatalError("")
      }
      activeFormat = selectedFormat

      if let preferredFps = preferredSpec.fps {
        activeVideoMinFrameDuration = CMTimeMake(1, preferredFps)
        activeVideoMaxFrameDuration = CMTimeMake(1, preferredFps)
        unlockForConfiguration()
      }
    }
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0"
toolsVersion="15505" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none"
useAutolayout="YES" useTraitCollections="YES" colorMatched="YES"
initialViewController="h72-q5-prT">
   <device id="retina4_7" orientation="portrait" appearance="light"/>
   <dependencies>
      <deployment identifier="iOS"/>
      <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="15510"/>
      <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
   </dependencies>
   <scenes>
      <!--Navigation Controller-->
      <scene sceneID="Dfj-YK-ZAS">
         <objects>
            <navigationController id="h72-q5-prT" sceneMemberID="viewController">
               <navigationBar key="navigationBar" contentMode="scaleToFill"
insetsLayoutMarginsFromSafeArea="NO" translucent="NO" id="hVX-SY-eYi">
                  <rect key="frame" x="0.0" y="0.0" width="375" height="44"/>
                  <autoresizingMask key="autoresizingMask"/>
                  <color key="tintColor" red="0.4392156862745098" green="0.99607843137254903"
blue="0.76078431372549016" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                  <color key="barTintColor" red="0.094117647058823528"
green="0.16470588235294117" blue="0.25882352941176473" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                  <textAttributes key="titleTextAttributes">
                     <fontDescription key="fontDescription" type="system" weight="medium"
pointSize="20"/>
                     <color key="textColor" red="0.5490196078431373"
green="0.96470588235294119" blue="0.99607843137254903" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                  </textAttributes>
               </navigationBar>
               <connections>
                  <segue destination="o4f-ub-ffb" kind="relationship"
relationship="rootViewController" id="hdM-xK-exT"/>
               </connections>
            </navigationController>
            <placeholder placeholderIdentifier="IBFirstResponder" id="TPN-3V-0eG"
userLabel="First Responder" sceneMemberID="firstResponder"/>
         </objects>
         <point key="canvasLocation" x="-1335" y="-57"/>
      </scene>
      <!--Start-->
      <scene sceneID="euJ-qH-Aw8">
         <objects>
            <viewController id="o4f-ub-ffb" customClass="StartViewController"
customModule="Driver_Assist" customModuleProvider="target"
sceneMemberID="viewController">
```

```
            <layoutGuides>
               <viewControllerLayoutGuide type="top" id="VTd-9h-bce"/>
               <viewControllerLayoutGuide type="bottom" id="Qwp-rr-u0N"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="fhU-XB-KeD">
               <rect key="frame" x="0.0" y="0.0" width="375" height="623"/>
               <autoresizingMask key="autoresizingMask" widthSizable="YES"
heightSizable="YES"/>
               <subviews>
                  <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Welcome to Magic
Vision!" textAlignment="center" lineBreakMode="tailTruncation" numberOfLines="2"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="PQb-Hw-pNe">
                     <rect key="frame" x="20" y="20" width="335" height="103"/>
                     <constraints>
                        <constraint firstAttribute="height" constant="103" id="aGI-q7-8hw"/>
                     </constraints>
                     <fontDescription key="fontDescription" type="system" weight="semibold"
pointSize="36"/>
                     <color key="textColor" red="1" green="0.83541802201146786"
blue="0.27215012216443435" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                     <nil key="highlightedColor"/>
                     <color key="shadowColor" red="0.99577091942148765" green="0.0"
blue="0.084133019106422832" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                     <size key="shadowOffset" width="1" height="1"/>
                  </label>
                  <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Developed by Konstantin
Stolyarenko" textAlignment="center" lineBreakMode="tailTruncation" numberOfLines="3"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="c2l-Sq-q2y">
                     <rect key="frame" x="20" y="553" width="335" height="50"/>
                     <constraints>
                        <constraint firstAttribute="height" constant="50" id="ge2-hy-5EN"/>
                     </constraints>
                     <fontDescription key="fontDescription" type="system" weight="semibold"
pointSize="20"/>
                     <color key="textColor" red="0.51014055369975042"
green="0.96639936779830971" blue="1" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                     <nil key="highlightedColor"/>
                  </label>
                  <button opaque="NO" contentMode="scaleToFill"
contentHorizontalAlignment="center" contentVerticalAlignment="center"
buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="V8o-2u-u0h">
                     <rect key="frame" x="140" y="182" width="95" height="30"/>
                     <constraints>
                        <constraint firstAttribute="height" constant="30" id="wUX-Gy-dSL"/>
                     </constraints>
                     <fontDescription key="fontDescription" type="system" weight="black"
pointSize="30"/>
```

```
                        <size key="titleShadowOffset" width="3" height="3"/>
                        <state key="normal" title="Vision">
                            <color key="titleColor" red="0.4392156862745098"
green="0.99607843137254903" blue="0.76078431372549016" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                            <color key="titleShadowColor" red="1" green="0.015699459194198573"
blue="0.13927185307131518" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                        </state>
                        <connections>
                            <action selector="startVision:" destination="o4f-ub-ffb"
eventType="touchUpInside" id="EwX-UU-mMX"/>
                        </connections>
                    </button>
                    <button opaque="NO" contentMode="scaleToFill"
contentHorizontalAlignment="center" contentVerticalAlignment="center"
buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="TW2-N3-H7y">
                        <rect key="frame" x="126" y="222" width="123" height="30"/>
                        <constraints>
                            <constraint firstAttribute="height" constant="30" id="Q06-qJ-7z6"/>
                            <constraint firstAttribute="height" relation="greaterThanOrEqual"
constant="30" id="RdT-bP-0Cb"/>
                        </constraints>
                        <fontDescription key="fontDescription" type="system" weight="black"
pointSize="30"/>
                        <size key="titleShadowOffset" width="2" height="2"/>
                        <state key="normal" title="Core ML">
                            <color key="titleColor" red="0.91459517045454541" green="0.0"
blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                            <color key="titleShadowColor" red="0.4392156862745098"
green="0.99607843137254903" blue="0.76078431372549016" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                        </state>
                        <connections>
                            <action selector="startML:" destination="o4f-ub-ffb"
eventType="touchUpInside" id="uza-ie-Dwt"/>
                        </connections>
                    </button>
                    <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Choose FrameWork"
textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines"
adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="JDM-7Q-
UIH">
                        <rect key="frame" x="112.5" y="141" width="150.5" height="21"/>
                        <constraints>
                            <constraint firstAttribute="height" constant="21" id="p4q-g0-9Up"/>
                        </constraints>
                        <fontDescription key="fontDescription" type="italicSystem" pointSize="17"/>
                        <color key="textColor" red="0.54117647058823526"
green="0.95294117647058818" blue="0.98431372549019602" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                        <nil key="highlightedColor"/>
                    </label>
```

```
                    <button opaque="NO" contentMode="scaleToFill"
contentHorizontalAlignment="center" contentVerticalAlignment="center"
buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="XV1-ot-6Ge">
                        <rect key="frame" x="90.5" y="292" width="194" height="48"/>
                        <fontDescription key="fontDescription" type="system" weight="black"
pointSize="30"/>
                        <size key="titleShadowOffset" width="2" height="2"/>
                        <state key="normal" title="Driver Assist">
                          <color key="titleColor" systemColor="systemTealColor"
red="0.35294117650000001" green="0.7843137255" blue="0.98039215690000003" alpha="1"
colorSpace="custom" customColorSpace="sRGB"/>
                          <color key="titleShadowColor" systemColor="systemIndigoColor"
red="0.34509803919999998" green="0.33725490200000002" blue="0.83921568629999999"
alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                        </state>
                        <connections>
                          <action selector="startDriverAssist:" destination="o4f-ub-ffb"
eventType="touchUpInside" id="h7y-gG-c0H"/>
                        </connections>
                    </button>
                </subviews>
                <color key="backgroundColor" red="0.086697183550000001"
green="0.16263428329999999" blue="0.26226973529999997" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                <constraints>
                    <constraint firstItem="JDM-7Q-UIH" firstAttribute="top" secondItem="PQb-Hw-
pNe" secondAttribute="bottom" constant="18" id="1sV-vT-BkE"/>
                    <constraint firstItem="Qwp-rr-u0N" firstAttribute="top" secondItem="c2l-Sq-
q2y" secondAttribute="bottom" constant="20" id="82C-Kl-SGJ"/>
                    <constraint firstItem="XV1-ot-6Ge" firstAttribute="top" secondItem="TW2-N3-
H7y" secondAttribute="bottom" constant="40" id="A2Y-3l-HDd"/>
                    <constraint firstItem="PQb-Hw-pNe" firstAttribute="leading" secondItem="fhU-
XB-KeD" secondAttribute="leading" constant="20" id="KHz-dR-mO7"/>
                    <constraint firstAttribute="trailing" secondItem="c2l-Sq-q2y"
secondAttribute="trailing" constant="20" id="PLk-z2-iIU"/>
                    <constraint firstItem="V8o-2u-u0h" firstAttribute="top" secondItem="JDM-7Q-
UIH" secondAttribute="bottom" constant="20" id="WxE-C6-Yia"/>
                    <constraint firstItem="V8o-2u-u0h" firstAttribute="centerX" secondItem="fhU-
XB-KeD" secondAttribute="centerX" id="ZAd-q4-Z0O"/>
                    <constraint firstItem="XV1-ot-6Ge" firstAttribute="centerX" secondItem="fhU-
XB-KeD" secondAttribute="centerX" id="ZbD-Wh-agG"/>
                    <constraint firstItem="c2l-Sq-q2y" firstAttribute="leading" secondItem="fhU-
XB-KeD" secondAttribute="leading" constant="20" id="bQg-DB-Q4i"/>
                    <constraint firstItem="TW2-N3-H7y" firstAttribute="centerX" secondItem="fhU-
XB-KeD" secondAttribute="centerX" id="dhT-ek-dCL"/>
                    <constraint firstItem="TW2-N3-H7y" firstAttribute="top" secondItem="V8o-2u-
u0h" secondAttribute="bottom" constant="10" id="hJz-ao-wfj"/>
                    <constraint firstAttribute="trailing" secondItem="PQb-Hw-pNe"
secondAttribute="trailing" constant="20" id="lhv-7g-n2V"/>
                    <constraint firstItem="PQb-Hw-pNe" firstAttribute="top" secondItem="VTd-9h-
bce" secondAttribute="bottom" constant="20" id="rdj-oE-SbY"/>
                    <constraint firstItem="TW2-N3-H7y" firstAttribute="centerX" secondItem="fhU-
```

```
XB-KeD" secondAttribute="centerX" id="upm-tq-dRT"/>
                    <constraint firstItem="JDM-7Q-UIH" firstAttribute="centerX" secondItem="fhU-
XB-KeD" secondAttribute="centerX" id="wfL-cQ-RM8"/>
                </constraints>
            </view>
            <navigationItem key="navigationItem" title="Start" id="Mtr-Li-c8D"/>
            <connections>
                <segue destination="0Xl-yk-nao" kind="show" identifier="openVisionML"
id="ILw-la-CZS"/>
                <segue destination="VRS-II-em1" kind="show" identifier="openDriverAssist"
id="bGQ-CX-X4j"/>
            </connections>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="Hf0-z3-xY4"
userLabel="First Responder" sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="-639.20000000000005" y="-57.121439280359823"/>
</scene>
<!--Vision ON-->
<scene sceneID="71d-m2-KJ4">
    <objects>
        <viewController title="Vision ON" id="0Xl-yk-nao"
customClass="MainViewController" customModule="Driver_Assist"
customModuleProvider="target" sceneMemberID="viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="Cze-I2-EGa"/>
                <viewControllerLayoutGuide type="bottom" id="Z5a-YL-kYA"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="f5H-zY-hKe">
                <rect key="frame" x="0.0" y="0.0" width="375" height="623"/>
                <autoresizingMask key="autoresizingMask" widthSizable="YES"
heightSizable="YES"/>
                <subviews>
                    <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConst   -
raints="NO" id="MPW-dG-rVE">
                        <rect key="frame" x="0.0" y="0.0" width="375" height="623"/>
                        <subviews>
                            <view hidden="YES" clipsSubviews="YES" contentMode="scaleToFill"
translatesAutoresizingMaskIntoConstraints="NO" id="75C-Ek-KZO">
                                <rect key="frame" x="20" y="40" width="335" height="349"/>
                                <color key="backgroundColor" white="1" alpha="1"
colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>
                                <constraints>
                                    <constraint firstAttribute="height" constant="349" id="bOV-KD-tpo"/>
                                </constraints>
                            </view>
                            <label hidden="YES" opaque="NO" userInteractionEnabled="NO"
contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Label"
textAlignment="center" lineBreakMode="tailTruncation" numberOfLines="0"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="Fr2-OW-lop">
                                <rect key="frame" x="20" y="397" width="335" height="30"/>
                                <color key="backgroundColor" red="0.098039215686274508"
```

```
green="0.16862745098039217" blue="0.25490196078431371" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                            <fontDescription key="fontDescription" type="system" pointSize="25"/>
                            <color key="textColor" red="0.0044076547677168731" green="1"
blue="0.12365967694519431" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                            <nil key="highlightedColor"/>
                        </label>
                    </subviews>
                    <color key="backgroundColor" red="0.094117647058823528"
green="0.16470588235294117" blue="0.25882352941176473" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                    <constraints>
                        <constraint firstItem="Fr2-OW-lop" firstAttribute="centerX"
secondItem="MPW-dG-rVE" secondAttribute="centerX" id="156-u8-nok"/>
                        <constraint firstItem="Fr2-OW-lop" firstAttribute="leading"
secondItem="MPW-dG-rVE" secondAttribute="leading" constant="20" id="E9R-El-nET"/>
                        <constraint firstItem="Fr2-OW-lop" firstAttribute="top" secondItem="75C-
Ek-KZO" secondAttribute="bottom" constant="8" id="FHh-Mt-Ycd"/>
                        <constraint firstItem="75C-Ek-KZO" firstAttribute="leading"
secondItem="MPW-dG-rVE" secondAttribute="leading" constant="20" id="Jx8-Hf-gEf"/>
                        <constraint firstItem="75C-Ek-KZO" firstAttribute="top"
secondItem="MPW-dG-rVE" secondAttribute="top" constant="40" id="LeA-wc-vff"/>
                        <constraint firstAttribute="trailing" secondItem="Fr2-OW-lop"
secondAttribute="trailing" constant="20" id="gkk-z4-RA4"/>
                        <constraint firstAttribute="trailing" secondItem="75C-Ek-KZO"
secondAttribute="trailing" constant="20" id="kQY-ff-4FJ"/>
                    </constraints>
                </view>
                <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConst   -
raints="NO" id="xc3-W5-qAq">
                    <rect key="frame" x="8" y="583" width="359" height="32"/>
                    <subviews>
                        <label opaque="NO" contentMode="left" horizontalHuggingPriority="251"
verticalHuggingPriority="251" text="" lineBreakMode="tailTruncation" numberOfLines="100"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="hpR-SF-YSn">
                            <rect key="frame" x="16" y="16" width="327" height="0.0"/>
                            <fontDescription key="fontDescription" name="Menlo-Bold"
family="Menlo" pointSize="16"/>
                            <color key="textColor" red="0.5490196078431373"
green="0.96470588235294119" blue="0.99607843137254903" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                            <nil key="highlightedColor"/>
                        </label>
                    </subviews>
                    <color key="backgroundColor" white="0.0" alpha="0.5"
colorSpace="calibratedWhite"/>
                    <constraints>
                        <constraint firstAttribute="bottom" secondItem="hpR-SF-YSn"
secondAttribute="bottom" constant="16" id="DvO-4t-Nir"/>
                        <constraint firstItem="hpR-SF-YSn" firstAttribute="leading"
secondItem="xc3-W5-qAq" secondAttribute="leading" constant="16" id="QeY-hJ-Cwm"/>
                        <constraint firstItem="hpR-SF-YSn" firstAttribute="top" secondItem="xc3-
```

```
W5-qAq" secondAttribute="top" constant="16" id="fRn-HZ-A7M"/>
                        <constraint firstAttribute="trailing" secondItem="hpR-SF-YSn"
secondAttribute="trailing" constant="16" id="odX-Yc-N24"/>
                    </constraints>
                    <userDefinedRuntimeAttributes>
                        <userDefinedRuntimeAttribute type="number"
keyPath="layer.cornerRadius">
                            <integer key="value" value="2"/>
                        </userDefinedRuntimeAttribute>
                    </userDefinedRuntimeAttributes>
                </view>
            </subviews>
            <color key="backgroundColor" white="1" alpha="1"
colorSpace="calibratedWhite"/>
            <constraints>
                <constraint firstAttribute="trailing" secondItem="xc3-W5-qAq"
secondAttribute="trailing" constant="8" id="GbP-lM-qpb"/>
                <constraint firstAttribute="trailing" secondItem="MPW-dG-rVE"
secondAttribute="trailing" id="GpE-Xt-H2b"/>
                <constraint firstItem="MPW-dG-rVE" firstAttribute="top" secondItem="Cze-I2-
EGa" secondAttribute="bottom" id="I48-Xy-1Mx"/>
                <constraint firstAttribute="bottom" secondItem="MPW-dG-rVE"
secondAttribute="bottom" id="N7z-Ew-IAo"/>
                <constraint firstItem="Z5a-YL-kYA" firstAttribute="top" secondItem="xc3-W5-
qAq" secondAttribute="bottom" constant="8" id="Q6h-tE-Uil"/>
                <constraint firstItem="MPW-dG-rVE" firstAttribute="leading" secondItem="f5H-
zY-hKe" secondAttribute="leading" id="kVi-v1-w6X"/>
                <constraint firstItem="xc3-W5-qAq" firstAttribute="leading" secondItem="f5H-
zY-hKe" secondAttribute="leading" constant="8" id="qBL-IY-3ly"/>
            </constraints>
        </view>
        <connections>
            <outlet property="focusedView" destination="75C-Ek-KZO" id="LBB-Fe-d9J"/>
            <outlet property="predictLabel" destination="hpR-SF-YSn" id="ac2-lY-Lsw"/>
            <outlet property="previewView" destination="MPW-dG-rVE" id="6ZR-uF-fDC"/>
            <outlet property="trustedLabel" destination="Fr2-OW-lop" id="hJF-4E-NYH"/>
        </connections>
    </viewController>
    <placeholder placeholderIdentifier="IBFirstResponder" id="8f5-Pe-2NG"
userLabel="First Responder" sceneMemberID="firstResponder"/>
  </objects>
  <point key="canvasLocation" x="36" y="-57.121439280359823"/>
</scene>
<!--Driver Assist View Controller-->
<scene sceneID="42J-2U-DNe">
  <objects>
    <viewController id="VRS-II-em1" customClass="DriverAssistViewController"
customModule="Driver_Assist" customModuleProvider="target"
sceneMemberID="viewController">
        <layoutGuides>
            <viewControllerLayoutGuide type="top" id="0mC-OX-b3S"/>
            <viewControllerLayoutGuide type="bottom" id="dU0-Qd-mhF"/>
        </layoutGuides>
```

```
            <view key="view" contentMode="scaleToFill" id="J0D-bL-aRb">
                <rect key="frame" x="0.0" y="0.0" width="375" height="623"/>
                <autoresizingMask key="autoresizingMask" widthSizable="YES"
heightSizable="YES"/>
                <subviews>
                    <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConst -
raints="NO" id="MHJ-M1-gsV" userLabel="Video Preview">
                        <rect key="frame" x="0.0" y="0.0" width="375" height="623"/>
                        <color key="backgroundColor" red="0.090520299969999996"
green="0.1573103964" blue="0.26275795699999999" alpha="1" colorSpace="custom"
customColorSpace="displayP3"/>
                    </view>
                    <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConst -
raints="NO" id="2hO-RR-r4v">
                        <rect key="frame" x="8" y="564" width="359" height="51"/>
                        <subviews>
                            <label opaque="NO" contentMode="left" horizontalHuggingPriority="251"
verticalHuggingPriority="251" text="rfr" lineBreakMode="tailTruncation" numberOfLines="100"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="3OZ-4m-BVI">
                                <rect key="frame" x="16" y="16" width="327" height="19"/>
                                <fontDescription key="fontDescription" name="Menlo-Bold"
family="Menlo" pointSize="16"/>
                                <color key="textColor" red="0.54901960780000003"
green="0.96470588239999999" blue="0.99607843139999996" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
                                <nil key="highlightedColor"/>
                            </label>
                        </subviews>
                        <color key="backgroundColor" white="0.0" alpha="0.5"
colorSpace="calibratedWhite"/>
                        <constraints>
                            <constraint firstItem="3OZ-4m-BVI" firstAttribute="top" secondItem="2hO-
RR-r4v" secondAttribute="top" constant="16" id="AwH-R4-Nrs"/>
                            <constraint firstAttribute="bottom" secondItem="3OZ-4m-BVI"
secondAttribute="bottom" constant="16" id="hlM-KK-aHB"/>
                            <constraint firstAttribute="trailing" secondItem="3OZ-4m-BVI"
secondAttribute="trailing" constant="16" id="rXd-0T-fLi"/>
                            <constraint firstItem="3OZ-4m-BVI" firstAttribute="leading"
secondItem="2hO-RR-r4v" secondAttribute="leading" constant="16" id="ymc-Xl-AIF"/>
                        </constraints>
                        <userDefinedRuntimeAttributes>
                            <userDefinedRuntimeAttribute type="number"
keyPath="layer.cornerRadius">
                                <integer key="value" value="2"/>
                            </userDefinedRuntimeAttribute>
                        </userDefinedRuntimeAttributes>
                    </view>
                </subviews>
                <color key="backgroundColor" white="0.0" alpha="1"
colorSpace="calibratedWhite"/>
                <constraints>
                    <constraint firstAttribute="bottom" secondItem="MHJ-M1-gsV"
```

```
secondAttribute="bottom" id="1CU-fg-IAn"/>
                        <constraint firstItem="dU0-Qd-mhF" firstAttribute="top" secondItem="MHJ-M1-
gsV" secondAttribute="bottom" id="93E-Ij-6Pi"/>
                        <constraint firstAttribute="trailing" secondItem="MHJ-M1-gsV"
secondAttribute="trailing" id="FEA-od-xER"/>
                        <constraint firstAttribute="trailing" secondItem="2hO-RR-r4v"
secondAttribute="trailing" constant="8" id="I7i-PZ-cne"/>
                        <constraint firstItem="MHJ-M1-gsV" firstAttribute="top" secondItem="J0D-bL-
aRb" secondAttribute="top" id="L1W-yN-6PA"/>
                        <constraint firstItem="dU0-Qd-mhF" firstAttribute="top" secondItem="2hO-RR-
r4v" secondAttribute="bottom" constant="8" id="XTV-rl-Zhz"/>
                        <constraint firstItem="MHJ-M1-gsV" firstAttribute="leading" secondItem="J0D-
bL-aRb" secondAttribute="leading" id="ctk-yj-6M8"/>
                        <constraint firstItem="MHJ-M1-gsV" firstAttribute="leading" secondItem="J0D-
bL-aRb" secondAttribute="leading" id="hMb-V5-gVF"/>
                        <constraint firstItem="MHJ-M1-gsV" firstAttribute="top" secondItem="0mC-
OX-b3S" secondAttribute="bottom" id="kMk-Ph-rmc"/>
                        <constraint firstAttribute="trailing" secondItem="MHJ-M1-gsV"
secondAttribute="trailing" id="qz4-Yv-yOi"/>
                        <constraint firstItem="2hO-RR-r4v" firstAttribute="leading" secondItem="J0D-
bL-aRb" secondAttribute="leading" constant="8" id="rMG-1f-9Uh"/>
                    </constraints>
                </view>
                <navigationItem key="navigationItem" id="OKZ-nE-XWQ"/>
                <connections>
                    <outlet property="timeLabel" destination="3OZ-4m-BVI" id="TVj-Eu-ldF"/>
                    <outlet property="videoPreview" destination="MHJ-M1-gsV" id="cKW-1R-c2D"/>
                </connections>
            </viewController>
            <placeholder placeholderIdentifier="IBFirstResponder" id="fmQ-tR-gAh"
sceneMemberID="firstResponder"/>
        </objects>
        <point key="canvasLocation" x="717.60000000000002" y="-57.121439280359823"/>
    </scene>
  </scenes>
</document>
```

ВІДОМІСТЬ АТЕСТАЦІЙНОЇ РОБОТИ

Позначення	Найменування	Дод. відомості
	Текстові документи	
1	Пояснювальна записка	96 с.
2	Презентаційний матеріал	28 с.
	Інші документи	
3	Роздруківки програм	32 с.
4	Рецензія	2 с.
5	Відгук керівника	1 с.

					Розпізнавання автомобілів в реальному часі за допомогою комп'ютерного зору і машинного навчання			
Змін	**Арк.**	**Номер докум.**	**Підп.**	**Дата**				
Розроб.		**Столяренко К.С.**			**(Тема роботи) Відомість атестаційної роботи**		**Аркуш**	**Аркушів**
Перевір.		**Кіріченко Л.О.**						
Н. контр.		**Сидоров М.В.**				**ХНУРЕ кафедра ПМ**		
Затв.		**Тевяшев А.Д.**						